Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 153 (2006) 161–175

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Towards a Logic for Performance and Mobility

Rocco De Nicola[a] Joost-Pieter Katoen[b] Diego Latella[c]
Mieke Massink[c]

[a] *Univ. of Firenze, Firenze, Italy*
denicola@dsi.unifi.it

[b] *RWTH Aachen, Aachen, Germany, and*
Univ. of Twente, Enschede, The Netherlands
katoen@cs.rwth-aachen.de

[c] *Istituto di Scienza e Tecnologie dell'Informazione, CNR, Pisa, Italy*
{diego.latella,mieke.massink}@isti.cnr.it

Abstract

Klaim is an experimental language designed for modeling and programming distributed systems composed of mobile components where distribution awareness and dynamic system architecture configuration are key issues. StocKlaim [13] is a Markovian extension of the core subset of Klaim which includes process distribution, process mobility, asynchronous communication, and site creation. In this paper, *MoSL*, a temporal logic for StocKlaim is proposed which addresses and integrates the issues of distribution awareness and mobility and those concerning stochastic behaviour of systems. The satisfiability relation is formally defined over labelled Markov chains. A large fragment of the proposed logic can be translated to action-based CSL for which efficient model-checkers exist. This way, such model-checkers can be used for the verification of StocKlaim models against *MoSL* properties. An example application is provided in the present paper.

Keywords: mobility, stochastic logics, process algebra

1 Introduction

During the last decades, computer systems have changed from isolated static devices to machines that are highly interconnected to perform tasks in a cooperative and coordinated manner. These modern, complex distributed systems—also known as *global or network-aware computers* [7]—are highly dynamic and have to deal with frequent changes of the network environment.

New features such as distribution awareness and code mobility play a prominent role in the concept of global computing. Dedicated programming and specification formalisms have been developed that can deal with issues such as (code and agent) mobility, remote execution, and privacy and security aspects (e.g., data integrity). Prominent examples of such languages and frameworks are, among others, Obliq [6], Seal [9], and Klaim [11].

Due to their enormous size—networks typically consist of thousands or even millions of nodes—and their strong reliance on mobility and interaction, performance and dependability issues of global computers are of utmost importance for "network-aware computing". Spontaneous computer crashes may easily lead to failure of remote execution or process movement, while spurious network hick ups may cause loss of code fragments or unpredictable delays. The presence of such random phenomena implies that correctness of practical global programs and their privacy guarantees are no longer rigid notions ("either it is safe or it is not"), but have a less absolute nature ("in 97% of the cases, privacy can be ensured"). The intrinsic complexity of global computers, though, complicates the assessment of these issues severely. Systematic methods, techniques and tools are therefore needed to establish performance and dependability requirements and guarantees.

This paper reports on our initial attempts towards such systematic methods. We consider an extension of the core subset of Klaim [4] with random delays, as proposed in [13]. This yields an integrated specification language supporting major global computing paradigms such as process mobility, process distribution, asynchronous communication of node names and processes through shared local repositories (i.e., tuple spaces), and dynamic node creation, as well as randomly delayed activities. For the sake of simplicity we restrict to exponential delays. This facilitates the use of existing numerical techniques and software tools. The generalization of our approach toward general distributions can be performed along the lines of [10].

The main contribution of this paper is a state and action-based, branching-time temporal logic that permits the specification of properties of stochastic core Klaim terms. The logic permits to consider the mobile and distributed aspects of global computing, its performance and dependability requirements, and their combination. It allows, for instance, to state that "in equilibrium, the probability that a piece of code currently at site l eventually ends up at site l' exceeds 0.9" and "a certain network configuration can be reached without ever dynamically creating a new site". The syntax and semantics of the logic are detailed. It is shown that a large fragment of the proposed logic can be mapped onto action-based CSL (aCSL [19]). Due to space limitations, the details of the mapping are not shown in the present paper. The interested reader is referred to [12] for the complete definition of the mapping as well as the formal proof of its correctness. Our mapping facilitates the use of the existing model checker ETMCC [20] for the automated verification of StocKlaim programs. ETMCC provides efficient algorithms to check an aCSL formula against a continuous time Markov chain (CTMC). It has been used for the validation of a number of real-life applications. We illustrate our approach by modelling the spreading of a virus through a network and verifying stochastic properties such as "the probability that the virus is spread to location l within t time-units is less than 10^{-4}". Several (temporal) logics have been proposed which aim at describing properties of systems related either to mobility ([3,14,5,8,16,22] among others) or to stochastic behaviour (e.g. [17,18,1,2,19]). To the best of our knowledge, the present paper is the first approach towards a probabilistic logic for mobility.

The paper is further organized as follows. Section 2 presents the StocKlaim language and its semantics in terms of action-labelled CTMCs. Section 3 introduces the "performobility" logic, its syntax and its semantics. The issues of translating the logic into aCSL are also briefly discussed. Section 4 presents the virus spreading case study and its verification results. Section 5 concludes the paper.

2 StocKlaim

In the following we briefly recall the StocKlaim language. In [13] the definition of the language is dealt with in full detail and a thorough discussion of the motivations for all design choices is presented. We refer the interested reader to the above mentioned paper.

Let $\mathcal{L}$, ranged over by $l, l', l_1, \ldots$, be a set of *localities*, $\mathcal{U}$, ranged over by $u, u', u_1, \ldots$, a set of *locality variables*, $\mathcal{A}$, ranged over by $A, A', A_1, \ldots$ a set of *process variables*, and $\mathcal{R}$, ranged over by $r, r', r_1, \ldots$ a set of *rate names.* We assume that the above sets are mutually disjoint. Moreover, we let ℓ, ℓ' range over $\mathcal{L} \cup \mathcal{U}$. The syntax of StocKlaim, is given in Table 1. A *network* node $l :: \langle l' \rangle$ intuitively models that value $\langle l' \rangle$ is stored, or located, in node, or locality, l. Similarly, for process P, $l :: \langle P \rangle$ means that P is stored in l as a piece of data[1]. On the other hand, $l :: P$ indicates that process P is running at locality l. Complex networks are built from simpler ones by means of the *network parallel operator* $\|$. Given network N, the set of values located at locality l consists of those l' and P such that $l :: \langle l' \rangle$ or $l :: \langle P \rangle$ occurs in N. The set of processes running at locality l coincides with all P

[1] Localities and processes are the only data available in the core language we consider in this paper.

N ::= Networks	P ::= Processes	a ::= Actions	T ::= Templates
$l :: \langle l' \rangle$	$\mathbf{nil}$	$\mathbf{out}\ \ell' @ \ell$	l
$\mid\ l :: \langle P \rangle$	$\mid\ (a, r).P$	$\mid\ \mathbf{out}\ P @ \ell$	$\mid\ !u$
$\mid\ l :: P$	$\mid\ P + P$	$\mid\ \mathbf{in}\ T @ \ell$	$\mid\ !A$
$\mid\ N \parallel N$	$\mid\ P \mid P$	$\mid\ \mathbf{eval}\ P @ \ell$	
	$\mid\ A$	$\mid\ \mathbf{newloc}\ u$	

Table 1
Syntax of StocKlaim

such that $l :: P$ occurs in N. The intuition behind the action prefix $(a, r).P$ is that the execution time of action a is exponentially distributed with rate specified by *rate name* r. Rate names are mapped to rate values by means of binding functions, which are (partial) functions from $\mathcal{R}$ to $\mathbb{R}^+$. Actions can be used for uploading (**out**) or downloading (**in**) data to or from specific localities. Such localities can also be expressed by means of variables. The only templates which can be used in download actions are locality *constants* or locality and process *variables*, that are prefixed by exclamation mark thus playing the role of parameters to be bound to values by the **in** action. Remote process evaluation (**eval**) is possible as well as the creation of new localities (**newloc**). Compound processes are built using the usual operators like choice $(+)$, interleaving $(|)$, and instantiation.

The operational semantics definition characterizes a transition relation over (network) configurations. A configuration is a triple (L, β, N)—henceforth also written as $L, \beta \vdash N$—where L is the finite set of localities occurring in N, $\beta : \mathcal{R} \mapsto \mathbb{R}^+$ is a mapping from rate names to rates, with $(dom\, \beta)$—the domain of β—finite, and N a network expression. We use L_c (resp. β_c and N_c) for L (resp. β and N) in configuration $c = L, \beta \vdash N$. Besides all the usual static semantics constraints (e.g. process names must be defined before usage and no variables must be free in the top-level network expression), we require that all rate names occurring in N are elements of $(dom\, \beta)$ and *distinct*. Rate name uniqueness ensures that whenever there is more than one way to perform the same action, the *total* rate for such action—i.e. that obtained considering the contribution of all the different ways to perform the action—will be taken into account, as discussed in [13].

We refer the reader to [12] for the formal definition of the *transition* relation $\rightarrow$. Here it suffices to say that the relation is essentially the same as in [13], with the only difference that every transition is now labeled not only with the rate name of the action which takes place by the transition, but also with the action itself as well as the locality where it is executed. For example, the rule for uploading (in node l) the piece of data l'' to node l' is the following one:

$$L, \beta \vdash l :: (\mathbf{out}\ l''\ @\ l', r).P \parallel l' :: P' \xrightarrow{(l, \mathbf{out}\ l''\ @\ l', r)} L, \beta \vdash l :: P \parallel l' :: P' \parallel l' :: \langle l'' \rangle$$

The action information in the transitions allows to refer to these activities in the logic. In order to apply the rules defining the transition relation, it is often necessary to rearrange configurations according to the structural congruence $\equiv$ defined in [13,12], which will be used also in the definition of the satisfaction relation of the logic. The laws which characterize such congruence include the usual axioms for associativity, commutativity and **nil**-neutrality for choice and parallel composition, KLAIM specific process *cloning*, and proper (rate name) *renaming*.

Let ACT denote the set of all *ground* actions (i.e. actions which do not contain variables) according to the syntax of Table 1. Let $(\mathsf{Loc}\ N)$ be the set of localities occurring in N. The operational semantics of a network specification (N, β) is defined, like in [13], as the labeled transition system (LTS) $TS(N, \beta) \stackrel{\text{def}}{=} (C, \Lambda, \rightarrow, c_0)$ where C is the set of (the representatives under $\equiv$ of) the configurations reachable from (the representative under $\equiv$ of) the initial state $c_0 \stackrel{\text{def}}{=} (\mathsf{Loc}\ N), \beta \vdash N$, via the transition relation $\rightarrow \subseteq C \times \Lambda \times C$ defined as mentioned above, with $\Lambda \subseteq \mathcal{L} \times ACT \times \mathcal{R}$.

The LTS associated to a network specification (N, β) can easily be translated to a CTMC, as we will show in the sequel. CTMCs have been extensively studied in the literature; a comprehensive treatment can be found in [21]. For the purposes of the present paper we will use the notion of *action-labelled CTMC*, as defined for instance in [19]:

Definition 2.1 An *action-labelled* CTMC (AMC) $\mathcal{M}$ is a triple $(S, Act, \longmapsto)$ where S is a set of states, Act is a set of actions, and $\longmapsto \ \subseteq S \times (Act \times \mathbb{R}^+) \times S$ is the transition relation.

In the sequel we consider only finite AMCs, i.e. finitely branching AMCs with a finite number of states. Moreover, Act, ranged over by $\gamma, \gamma', \gamma_1$, is a set of *located* actions (l, a) where a is a ground action and l the locality where a takes place (we omit the word "located" when this will not cause confusion). Transition $s \xmapsto{\gamma,\lambda} s'$ means that the process can move from state s to state s' while performing action γ with an execution time determined by an exponential distribution with rate λ. We say that $s \in S$ is *absorbing* if and only if there is no s', γ and λ such that $s \xmapsto{\gamma,\lambda} s'$. Notice that according to the above definition AMCs can have self-loops. The presence of such transitions does not affect standard CTMC measures and allows for a natural definition of the action-based temporal operators in the logic. The AMC associated to a

StocKlaim network specification is defined below, where the LTS associated to the specification is assumed to be finite, i.e. it is finitely branching and has a finite number of states[2].

Definition 2.2 Given a network specification (N, β) with $TS(N, \beta) = (C, \Lambda, \rightarrow, c_0)$ finite, the AMC $(S, Act, \longmapsto)$ associated to the specification and denoted by $AMC(N, \beta)$, is such that $S = C$, $Act \subseteq (\mathcal{L} \times ACT) \times \mathbb{R}^+$ and for all $c, c' \in S$, $c \xmapsto{(l,a),\lambda} c'$ if and only if $0 \neq \lambda = \sum_{c \xrightarrow{(l,a,r)} c'} (\beta_c\, r)$.

We close this section with the definition of *paths* which we will use in the next section. Given AMC $(S, Act, \longmapsto)$, $\Gamma \subseteq Act$, and $s, s' \in S$, let $\mathbf{R}_\Gamma(s, s')$, be defined as follows: $\mathbf{R}_\Gamma(s, s') \overset{\text{def}}{=} \sum_{\gamma\in\Gamma}\{\lambda \mid s \xmapsto{\gamma,\lambda} s'\}$ The definition of (maximal) *Paths* over an AMC follows:

Definition 2.3 Given AMC $\mathcal{M} = (S, Act, \longmapsto)$, an infinite *path* σ over $\mathcal{M}$ is a sequence $s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)s_2(\gamma_2, t_2)\ldots$ such that, for all $i \geqslant 0$, $s_i \in S, \gamma_i \in Act$, $t_i \in \mathbb{R}^+$ and $\mathbf{R}_{\{\gamma_i\}}(s_i, s_{i+1}) > 0$. For $i \geqslant 0$, we let $\sigma[i] = s_i$, $\alpha(\sigma, i) = \gamma_i$, and $\delta(\sigma, i) = t_i$. For $t \in \mathbb{R}_{\geqslant 0}$ and i the smallest index such that $t \leqslant \sum_{j=0}^{i} t_j$ we let $\sigma(t) = \sigma[i]$. Finite paths are defined similarly; the ending state is an absorbing state.

We let $\mathsf{len}\,\sigma$ denote the length of path σ and define it to be ∞ when σ is infinite. For any state s of an AMC $\mathcal{M}$, let $\mathsf{Paths}(s)$ denote the set of *all* (finite and infinite) paths $s_0(\gamma_0, t_0)s_1(\gamma_1, t_1)s_2(\gamma_2, t_2)\ldots$ over $\mathcal{M}$ with $s_0 = s$. A Borel space can be defined over $\mathsf{Paths}(s)$, together with its associated probability measure Pr, which is a slight extension of that defined in [2] in order to take both states and actions into consideration [12].

3 *MoSL*: a logic for StocKlaim

In this section, we present *MoSL*, a logic for the integrated specification of *functional*—i.e. qualitative—as well as *quantitative*—e.g. performance—properties of *mobile* systems modeled using StocKlaim. We also briefly discuss the translation of the logic to aCSL.

[2] There are several ways for assuring finiteness of the LTSs that are automatically generated from higher level specifications, like process algebras. Some rely on syntactical restrictions, like avoiding certain (combinations of) operators. Others, typically used in the context of verification tools design, are based on the introduction of constraints on certain kinds of resources.

Φ ::=	STATE FORMULAE	$\aleph$ ::=	ATOMIC PROPOSITIONS	ξ ::=	ACTION FORMULAE
	tt		$A@\ell$	$\mid$	tt
$\mid$	$\aleph$	$\mid$	$\langle\ell\rangle@\ell$	$\mid$	$\neg\,\xi$
$\mid$	$\neg\,\Phi$	$\mid$	$\langle A\rangle@\ell$	$\mid$	$\xi \vee \xi$
$\mid$	$\Phi \vee \Phi$			$\mid$	$\ell : \mathbf{out}\ \ell\ @\ \ell$
$\mid$	$\mathcal{S}_{\bowtie p}(\Phi)$	φ ::=	PATH FORMULAE	$\mid$	$\ell : \mathbf{out}\ A\ @\ \ell$
$\mid$	$\mathcal{P}_{\bowtie p}(\varphi)$		$\Phi\,{}_{\xi}\mathbf{U}^{<t}_{\xi}\,\Phi$	$\mid$	$\ell : \mathbf{in}\ \ell\ @\ \ell$
				$\mid$	$\ell : \mathbf{in}\ A\ @\ \ell$
		$\mid$	$\Phi\,{}_{\xi}\mathbf{U}^{<t}\,\Phi$	$\mid$	$\ell : \mathbf{eval}\ A\ @\ \ell$
				$\mid$	$\ell : \mathbf{newloc}\ u$

Table 2
Syntax of *MoSL*

3.1 Syntax of *MoSL*

Let $\mathcal{L}$, $\mathcal{U}$, and $\mathcal{A}$ be defined as in Sect. 2, $p \in [0,1]$ a probability value, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, and $t \in \mathbb{R}^+ \cup \{\infty\}$. The syntax of *MoSL* is defined in Table 2. The main difference between *MoSL* and aCSL is that besides the trivial proposition tt, state formulae include those built from $A@l$ (resp. $\langle l'\rangle@l$, $\langle A\rangle@l$) modeling the fact that process A is executing at locality l (resp. value l' or process A is stored at locality l). Locality variables will be bound by means of action formulae of the form ℓ : **newloc** u, as we will see later. Moreover, requirements on actions are expressed as boolean combinations of action propositions directly reflecting the actions of StocKlaim processesw and not as sets of symbolic actions. For instance, under the assumption that A is a specific process (i.e. there is a definition for A), action formula l : **in** A @ l' states that in locality l an input action is being performed by means of which A is downloaded from repository l'.

We recall from [1,2] that $\mathcal{P}_{\bowtie p}(\varphi)$ asserts that the probability measure of the set of paths satisfying φ meets the bound $\bowtie p$ while $\mathcal{S}_{\bowtie p}(\Phi)$ means that the steady-state probability for the set of states satisfying Φ meets the bound $\bowtie p$. The path formula $\Phi\,{}_{\xi}\mathbf{U}^{<t}\,\Phi'$ is fulfilled by a path if a Φ'-state is eventually reached by passing only through Φ-states before, while taking only transitions whose actions satisfy ξ; moreover, the Φ'-state should be reached within t time units. The formula $\Phi\,{}_{\xi}\mathbf{U}^{<t}_{\xi'}\,\Phi'$ requires moreover that (i) a move to a Φ'-state is actually made and that (ii) this transition is labeled by an action satisfying ξ'.

As it is clear from the syntactical definition of *MoSL*, formulae may contain both locality variables and process variables. While process variables

$\mathsf{Fr}(\mathrm{tt})$	$\overset{\mathrm{def}}{=} \varnothing$	$\mathsf{Fr}(\ell : \mathbf{out}\ \ell' \ @\ \ell'')$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell) \cup \mathsf{Fr}(\ell') \cup \mathsf{Fr}(\ell'')$
$\mathsf{Fr}(A@\ell)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell)$	$\mathsf{Fr}(\ell : \mathbf{out}\ A \ @\ \ell')$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell) \cup \mathsf{Fr}(\ell')$
$\mathsf{Fr}(\langle \ell' \rangle @\ell)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell') \cup \mathsf{Fr}(\ell)$	$\mathsf{Fr}(\ell : \mathbf{in}\ \ell' \ @\ \ell'')$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell) \cup \mathsf{Fr}(\ell') \cup \mathsf{Fr}(\ell'')$
$\mathsf{Fr}(\langle A \rangle @\ell)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell)$	$\mathsf{Fr}(\ell : \mathbf{in}\ A \ @\ \ell')$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell) \cup \mathsf{Fr}(\ell')$
$\mathsf{Fr}(\neg\, \Phi)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\Phi)$	$\mathsf{Fr}(\ell : \mathbf{eval}\ A \ @\ \ell')$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell) \cup \mathsf{Fr}(\ell')$
$\mathsf{Fr}(\Phi \vee \Psi)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\Phi) \cup \mathsf{Fr}(\Psi)$	$\mathsf{Fr}(\ell : \mathbf{newloc}\, u)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\ell)$
$\mathsf{Fr}(\mathcal{S}_{\bowtie p}(\Phi))$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\Phi)$	$\mathsf{Fr}(\Phi\, {}_{\xi}\mathbf{U}^{<t}_{\eta}\, \Psi)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\Phi) \cup \mathsf{Fr}(\xi) \cup \mathsf{Fr}(\eta) \cup$
$\mathsf{Fr}(\mathcal{P}_{\bowtie p}(\varphi))$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\varphi)$		$\mathsf{Fr}(\Psi) \setminus (\mathsf{Bnd}(\xi) \cup \mathsf{Bnd}(\eta))$
$\mathsf{Fr}(\xi \vee \eta)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\xi) \cup \mathsf{Fr}(\eta)$	$\mathsf{Fr}(\Phi\, {}_{\xi}\mathbf{U}^{<t}\, \Psi)$	$\overset{\mathrm{def}}{=} \mathsf{Fr}(\Phi) \cup \mathsf{Fr}(\xi) \cup (\mathsf{Fr}(\Psi) \setminus (\mathsf{Bnd}(\xi)))$

where we let $\mathsf{Fr}(l) \overset{\mathrm{def}}{=} \varnothing$ for all localities $l \in \mathcal{L}$, $\mathsf{Fr}(u) \overset{\mathrm{def}}{=} \{u\}$ for all locality *variables* $u \in \mathcal{U}$. Function Bnd is such that $\mathsf{Bnd}(\ell : \mathbf{newloc}\ u) \overset{\mathrm{def}}{=} \{u\}, \mathsf{Bnd}(\xi \vee \eta) \overset{\mathrm{def}}{=} \mathsf{Bnd}(\xi) \cap \mathsf{Bnd}(\eta)$, and $\mathsf{Bnd}(\xi) \overset{\mathrm{def}}{=} \varnothing$ in all other cases.

Fig. 1. Free locality variables in *MoSL*

will act as non-interpreted symbols in *MoSL* semantics, an action formula like l : **newloc** u may bind the locality variable u, when interpreted over an action of a path. Function Fr, defined in Fig.1, characterizes the set of free (locality) variables of *MoSL* formulae.
In the following only *well-formed* formulae will be considered. A formula Φ is well-formed if and only if it contains no free variables (i.e. $\mathsf{Fr}(\Phi) = \varnothing$).

A few comments on the definition of functions Fr and Bnd are in order. First, notice that free occurrences of u in Φ remain free also in $\Phi\, {}_{\xi}\mathbf{U}^{<t}_{\eta}\, \Psi$ (and $\Phi\, {}_{\xi}\mathbf{U}^{<t}\, \Psi$), *even* if they are bound by ξ (i.e. they are elements of $\mathsf{Bnd}(\xi)$). As we will see in the definition of the semantics of *MoSL*, a path formula of the logic will be interpreted, via relation $\models$, over a path σ. Now, for defining $\sigma \models \Phi\, {}_{\xi}\mathbf{U}^{<t}\, \Psi$ it is necessary to know if $\sigma[0] \models \Phi$; on the other hand, any binding of u by means of a sub-formula of ξ of the form ℓ : **newloc** u will take effect *only* from $\sigma[1]$ on, i.e. after the "first transition" has taken place. Similarly, we get $u \in \mathsf{Fr}(\Phi\, {}_{\xi}\mathbf{U}^{<t}_{\eta}\, \Psi)$ if $u \in \mathsf{Fr}(\eta)$, even in the case $u \in \mathsf{Bnd}(\xi)$. Another issue which deserves some attention is the definition of $\mathsf{Bnd}(\xi \vee \eta)$. Consider the formula $\mathrm{tt}\, {}_{\mathrm{tt}}\mathbf{U}^{<\infty}_{\xi}\, \Phi$, where ξ is the formula $(l$: **newloc** $u) \vee (l$: **out** l @ $l)$ which informally means that a state where Φ holds will eventually be reached via a transition corresponding to the execution of action **newloc** u or **out** l @ l (in locality l); in the latter case, no binding occurs for u and this would make it wrong to use the variable in Φ. Consequently, in such cases Bnd returns the empty set. Conversely, u can safely occur in Φ in the above formula if ξ stands for l_1 : **newloc** $u \vee l_2$: **newloc** u. Finally, notice that $\mathsf{Bnd}(\neg\, \xi)$ is the empty set since variables u occurring in formulae like

l : **newloc** u in ξ are actually *not* bound if $\neg\,\xi$ is satisfied. Strictly speaking, if for instance $\neg l$: **newloc** u is satisfied, then u should be bound to the locality l' appearing in the transition. A proper study of the contexts where the binding should take place and those in which this is not the case could allow for a more refined definition of Bnd. Another issue could be the requirement of $\mathsf{Bnd}(\xi) = \mathsf{Bnd}(\eta)$ for all action formulae $\xi \vee \eta$. This requirement forbids for instance the formula l : **newloc** $u_1 \vee l$: **newloc** u_2. A possible alternative could be to drop the requirement, defining $\mathsf{Bnd}(\xi \vee \eta) \overset{\text{def}}{=}$ if $(\mathsf{Bnd}(\xi) \neq \varnothing) \wedge (\mathsf{Bnd}(\eta) \neq \varnothing)$ then $(\mathsf{Bnd}(\xi) \cup \mathsf{Bnd}(\eta))$ else $\varnothing$. At the semantics level, when the formula is matched against l : **newloc** l' the substitution $[l'/u_1, l'/u_2]$ will be generated. On the other hand, one should still take care that formulae like l_1 : **newloc** $u_1 \vee l_2$: **newloc** u_2 are not allowed since whatever substitution will be generated at the semantics level, this would bind at most one of u_1 or u_2. In the present paper we adopt simpler, but safe solutions for both issues.

3.2 Semantics of *MoSL*

State formulae are interpreted over the states of $AMC(N,\beta) = (S, Act, \longmapsto\ )$ obtained from a StocKlaim network specification (N,β) as described in Sect. 2. Recall that the states of such AMC are StocKlaim configurations. *MoSL* formulae may contain locality variables which are instantiated by proper *substitutions* as described in the definition of the satisfaction relation below. Such substitutions are generated whenever an action $(l, \textbf{newloc}\ l')$ in a path of $AMC(N,\beta)$ satisfies an action formula of the form l : **newloc** u, for some $u \in \mathcal{U}$. The associated substitution is $[l'/u]$. By $\Phi[l_1/u_1, \ldots, l_n/u_n]$ we denote the formula obtained from Φ by replacing all free locality variables $u_1, \ldots, u_n$ of Φ by the corresponding localities $l_1, \ldots, l_n$. Let $[]$ denote the empty substitution and, without loss of generality, for $\Theta_1 = [l_1/u_1, \ldots, l_n/u_n, l'_1/u'_1, \ldots, l'_m/u'_m]$ and $\Theta_2 = [l''_1/u'_1, \ldots, l''_m/u'_m, l''_{m+1}/u'_{m+1}, \ldots l''_{m+h}/u'_{m+h}]$, with $\{u'_{m+1}, \ldots, u'_{m+h}\} \cap \{u_1, \ldots, u_n\} = \varnothing$ let $\Theta_1 \lhd \Theta_2$ be defined as the substitution

$$[l_1/u_1, \ldots, l_n/u_n, l''_1/u'_1, \ldots, l''_m/u'_m, l''_{m+1}/u'_{m+1}, \ldots, l''_{m+h}/u'_{m+h}]$$

The satisfaction relation for *state-formulae* is given in Table 3 and is self-explanatory. The definition of the satisfaction relation for *path-formulae* of the form $\Phi\,{}_{\xi}\mathbf{U}^{<t}\,\Psi$ is given in Table 4. The satisfaction relation for $\Phi\,{}_{\xi}\mathbf{U}^{<t}_{\eta}\,\Psi$ can be found in [12]; the only difference between such a formula and $\Phi\,{}_{\xi}\mathbf{U}^{<t}\,\Psi$ is that the Ψ-state *must* be reached by a η-transition (and consequently it cannot be the initial state). Table 4 makes use of the substitution generator function SBS, where $\mathsf{SBS}((l, \textbf{newloc}\ l'), l : \textbf{newloc}\ u) \overset{\text{def}}{=} [l'/u]$, $\mathsf{SBS}(\gamma, \xi \vee \eta) \overset{\text{def}}{=} \mathsf{SBS}(\gamma,\xi) \lhd \mathsf{SBS}(\gamma,\eta)$, and $\mathsf{SBS}(\gamma,\xi) \overset{\text{def}}{=} []$ in all other cases. Note that the well-

$s \models \mathrm{tt}$			$s \models A@l$	iff	$N_s \equiv N' \parallel l :: A$
$s \models \neg\Phi$	iff	not $s \models \Phi$	$s \models \langle l' \rangle @l$	iff	$N_s \equiv N' \parallel l :: \langle l' \rangle$
$s \models \Phi \vee \Psi$	iff	$s \models \Phi$ or $s \models \Psi$	$s \models \langle A \rangle @l$	iff	$N_s \equiv N' \parallel l :: \langle A \rangle$
$s \models \mathcal{S}_{\bowtie p}(\Phi)$	iff	$\lim_{t\to\infty} \Pr\{\sigma \in \mathit{Paths}(s) \mid \sigma(t) \models \Phi\} \bowtie p$			
$s \models \mathcal{P}_{\bowtie p}(\varphi)$	iff	$\Pr\{\sigma \in \mathit{Paths}(s) \mid \sigma \models \varphi\} \bowtie p$			

Table 3
Satisfaction relation for state-formulae.

$\sigma \models \Phi\, {}_{\xi}\mathbf{U}^{<t}\, \Psi$ iff $\sigma[0] \models \Psi$ or there exists $k : 0 < k \leqslant (\mathsf{len}\sigma)$ such that
the following three conditions hold:
- $\sigma[k] \models \Psi\Theta_k$
- $t > \sum_{i=0}^{k-1} \delta(\sigma, i)$
- $\sigma[i] \models \Phi\Theta_i$ and $\alpha(\sigma, i) \models \xi\Theta_i$, for all i with $0 \leqslant i < k$

where $\Theta_0 \stackrel{\mathrm{def}}{=} []$
$\Theta_i \stackrel{\mathrm{def}}{=} \Theta_{i-1} \triangleleft \mathsf{SBS}(\alpha(\sigma, i-1), \xi)$ for all $i > 0$

Table 4
Satisfaction relation for path-formulae.

$\gamma \models \mathrm{tt}$		$\gamma \models l : \mathbf{in}\ l' \ @\ l''$	iff $\gamma = (l, \mathbf{in}\ l' \ @\ l'')$
$\gamma \models \neg\xi$	iff not $\gamma \models \xi$	$\gamma \models l : \mathbf{in}\ A \ @\ l'$	iff $\gamma = (l, \mathbf{in}\ A \ @\ l')$
$\gamma \models \xi \vee \eta$	iff $\gamma \models \xi$ or $\gamma \models \eta$	$\gamma \models l : \mathbf{eval}\ A \ @\ l'$	iff $\gamma = (l, \mathbf{eval}\ A \ @\ l')$
$\gamma \models l : \mathbf{out}\ l' \ @\ l''$	iff $\gamma = (l, \mathbf{out}\ l' \ @\ l'')$	$\gamma \models l : \mathbf{newloc}\ u$	iff there exists $l' \in \mathcal{L}$ s.t. $\gamma = (l, \mathbf{newloc}\ l')$
$\gamma \models l : \mathbf{out}\ A \ @\ l'$	iff $\gamma = (l, \mathbf{out}\ A \ @\ l')$		

Table 5
Satisfaction relation for action-formulae.

formedness of (action) formulae guarantees that SBS generates a substitution, which binds at most one locality variable. As it should be clear from the formal definition, as soon as an action in a path satisfies an action formula of the form **newloc** u, variable u is properly bound and the substitution thus generated is used for binding the free variables in the scope of the action formula. The interpretation of action formulae is given in Table 5. The operators of $MoSL$ can be used for defining some useful derived operators e.g. $\Phi\, {}_{\xi}\mathbf{U}_{\eta}\, \Psi \stackrel{\mathrm{def}}{=} \Phi\, {}_{\xi}\mathbf{U}_{\eta}^{<\infty}\, \Psi$, $\mathbf{X}_{\xi}{}^{<t}\, \Phi \stackrel{\mathrm{def}}{=} \mathrm{tt}\, {}_{\mathrm{ff}}\mathbf{U}_{\xi}^{<t}\, \Phi$, or $\mathcal{P}_{\bowtie p}(\, {}_{A}\Diamond_{B}^{<t}\, \Phi) \stackrel{\mathrm{def}}{=} \mathcal{P}_{\bowtie p}(\mathrm{tt}\, {}_{A}\mathbf{U}_{B}^{<t}\, \Phi)$ besides the usual

ones like $\vee$, $\Rightarrow$, etc.[3]. In [12] a translation T from a large fragment of $MoSL$ to aCSL is presented together with a transformation of the AMC model to be checked, for network specification (N, β). A formal proof of their correctness is provided as well. The fragment $MoSL^-$ includes all $MoSL$ formulae except those which contain sub-formulae of the form ℓ : **newloc** u. The logic aCSL does not provide atomic propositions different from tt. Consequently, given a $MoSL$ formula Φ, the model transformation consists in *enriching* the labels of all those transitions emanating from any state s of $AMC(N, \beta)$ with (a coding of) the information on which atomic propositions occurring in Φ are satisfied by s, and which ones are not; we let $FAMC(N, \beta)$ denote such a transformed AMC. Similarly $\mathsf{T}(\Phi)$ will be constructed by moving all information on states—expressed by (combinations of) atomic propositions—forward to the index-labels of until operators. Notice that the translation is linear in the size of Φ. The correctness of the translation is shown by the following theorem, proved in [12], where $\mathcal{M}, s \models_{\mathcal{L}} \mathcal{F}$ means that state s of AMC $\mathcal{M}$ satisfies formula $\mathcal{F}$ of logic $\mathcal{L}$.

Theorem 3.1
For $MoSL^-$ formula Φ, and a network specification (N, β), the following holds: for all states s of $AMC(N, \beta)$, $AMC(N, \beta), s \models_{MoSL^-} \Phi$ if and only if $FAMC(N, \beta), s \models_{aCSL} \mathsf{T}(\Phi)$. □

4 Modeling and analysis of the spreading of a virus

In this section we show how StocKlaim can be used for modeling the *spreading* of a virus in a network. We also give examples of interesting qualitative and quantitative properties of the model that can be expressed in $MoSL$. This example has been inspired by a similar one in [15] and used also in [13].

We model a network as a set of nodes and the virus running on a node can move arbitrarily from the current node to a subset of adjacent nodes, infecting them. At each node, an operating system runs, which upon receiving the virus, can either run it or suppress it. In this paper, for the sake of simplicity we consider simple networks which are in fact grids of $n \times m$ nodes. Each node is connected with its four neighbors (north, south, east, west), except for border nodes, which lack some connections in the obvious way (e.g. the nodes on the east border have no east connection). Moreover, we assume that the virus can move only to *one* adjacent node. Finally, we refrain from modeling aspects of the virus other than the way it replicates in the network. In particular we do not consider the local effects of the virus and we let the virus die as soon as

[3] See [12] for the complete set of derived operators.

$$\begin{array}{lll}
\mathtt{V}_{ij} & \triangleq & (\mathbf{out}\ \mathtt{V}_{i-1j} \ @\ \mathtt{l}_{i-1j}, \mathtt{n}_{ij}).\mathbf{nil}\ + \\
 & & \text{/* alternative present only for } i > 1 \text{ */} \\
 & & (\mathbf{out}\ \mathtt{V}_{i+1j} \ @\ \mathtt{l}_{i+1j}, \mathtt{s}_{ij}).\mathbf{nil}\ + \\
 & & \text{/* alternative present only for } i < n \text{ */} \\
 & & (\mathbf{out}\ \mathtt{V}_{ij+1} \ @\ \mathtt{l}_{ij+1}, \mathtt{e}_{ij}).\mathbf{nil}\ + \\
 & & \text{/* alternative present only for } j < m \text{ */} \\
 & & (\mathbf{out}\ \mathtt{V}_{ij-1} \ @\ \mathtt{l}_{ij-1}, \mathtt{w}_{ij}).\mathbf{nil} \\
 & & \text{/* alternative present only for } j > 1 \text{ */} \\
 & & \\
\mathtt{O}_{ij} & \triangleq & (\mathbf{in}\ !\mathtt{C} \ @\ \mathtt{l}_{ij}, \mathtt{u}_{ij}).\mathtt{X}_{ij}\ + \\
 & & \text{/* the received virus is undetected and will run */} \\
 & & (\mathbf{in}\ !\mathtt{C} \ @\ \mathtt{l}_{ij}, \mathtt{d}_{ij}).\mathtt{O}_{ij} \\
 & & \text{/* the received virus is detected and suppressed */} \\
\mathtt{X}_{ij} & \triangleq & (\mathbf{eval}\ \mathtt{C} \ @\ \mathtt{l}_{ij}, \mathtt{r}_{ij}).\mathtt{O}_{ij} \\
 & & \text{/* the virus is activated */}
\end{array}$$

Fig. 2. Specification of an infected network

it has infected one of the neighbors of its locality. The specification scheme of the virus and the operating system running at each node is given in Fig. 2, where a network is conventionally represented as a $n \times m$ matrix of localities l_{ij}. For the verification, we chose $n = m = 3$ with the following initial state N_0: $\mathtt{l}_{11}{::}\mathtt{O}_{11} \parallel \mathtt{l}_{11}{::}\langle \mathtt{V}_{11} \rangle$, while $\mathtt{l}_{ij}{::}\mathtt{O}_{ij}$ for $1 \leqslant i, j \leqslant 3$ with $i \neq 1$ or $j \neq 1$. The resulting AMC is not shown for space reasons; it consists of 28 states and 52 transitions.

There are several interesting issues of the spreading of the virus which can be addressed using *MoSL*. The first property is an example of a purely state-based quantitative property. The probability that the virus is running at node $\mathtt{l}_{ij}$ within t time-units after the infection of node $\mathtt{l}_{11}$ is smaller than a given upper bound p. This property becomes more interesting when we define the rates associated to the detection (resp. lack of detection) of the virus in such a way that the operating systems of the localities on the diagonal from bottom-left to top-right—$\mathtt{O}_{31}$, $\mathtt{O}_{22}$, and $\mathtt{O}_{13}$—have a relatively high rate of detection and can be considered as a firewall to protect the nodes $\mathtt{l}_{32}$, $\mathtt{l}_{33}$, and $\mathtt{l}_{23}$. The property can be expressed in *MoSL* for locality $\mathtt{l}_{33}$ and $p = 0.2$ as $\mathcal{P}_{\leqslant 0.2}(\neg(\mathtt{V}_{33}@\mathtt{l}_{33})\ {}_{\mathtt{tt}}\mathbf{U}^{<t}\ \mathtt{V}_{33}@\mathtt{l}_{33})$. Fig. 3 shows the probability to reach, from the initial state, a state where the virus is running in locality $\mathtt{l}_{33}$. The measure is presented for time values ranging from 1 to 10 with $\beta_0\, e_{ij} = \beta_0\, n_{ij} = \beta_0\, s_{ij} = \beta_0\, w_{ij} = \beta_0\, r_{ij} = 2$ for $1 \leqslant i, j \leqslant 3$, $\beta_0\, d_{31} = \beta_0\, d_{22} = \beta_0\, d_{13} = 10$, and $\beta_0\, d_{ij} = 1$ otherwise, $\beta_0\, u_{31} = \beta_0\, u_{22} = \beta_0\, u_{13} = 1$, and $\beta_0\, u_{ij} = 10$ otherwise. The results have been computed by providing ETMCC [20] with the aCSL translation of the formula and corresponding AMC. We performed similar analyzes for different values of the detection

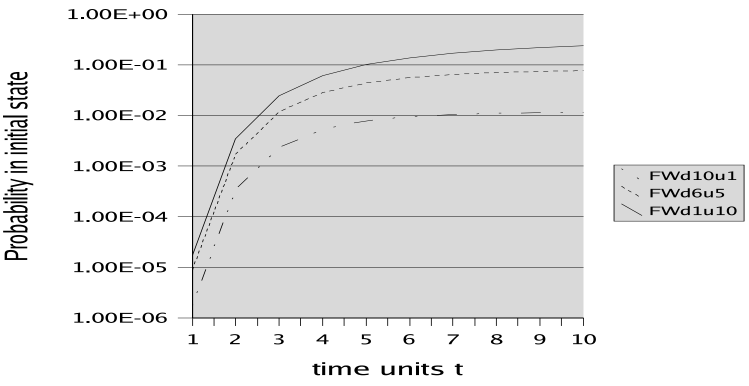

Fig. 3. Results for Firewalls with different detection capability

(resp. lack of detection) rates of the firewall. In particular for d_{31}, d_{22}, d_{13} and u_{31}, u_{22}, u_{13} ranging over $[1, \ldots, 10]$, with $d_{(4-i)i} + u_{(4-i)i}$ constant for $1 \leqslant i \leqslant 3$ (and equal to 11). For the sake of readability, in Fig. 3 we show the results only for $d_{31}, d_{22}, d_{13} \in \{1, 6, 10\}$ and $u_{31}, u_{22}, u_{13} \in \{1, 5, 10\}$. The results clearly indicate that for high detection rates the probability for locality l_{33} to run the virus within a certain time interval is lower.

Stochastic model-checking permits the verification of qualitative properties as a degenerate case of quantitative ones. For instance, an interesting property is: "Whenever a node is infected (i.e. a virus runs on it) the virus may move to a neighbor in the next step". For instance, in the case of node $\mathtt{l}_{33}$ the property of interest is $\mathtt{V}_{33}@\mathtt{l}_{33} \Rightarrow \langle \mathtt{l}_{33} : \textbf{out}\ \mathtt{V}_{32}\ @\ \mathtt{l}_{32} \rangle\, \mathrm{tt}$. The model-checker shows that this property holds in every state.

5 Conclusions and future work

In this paper *MoSL*, a stochastic logic for StocKlaim has been proposed. StocKlaim is a stochastic extension of the core subset of Klaim, a prototype language for modelling and programming global or network-aware computers. StocKlaim addresses process mobility, process distribution, asynchronous communication through shared local repositories (i.e., tuple spaces), and dynamic node creation, as well as randomly delayed activities.

The logic addresses both spatial and temporal notions to reflect both the topological structure of systems and their evolution over time. In connection with the duration attributes of StocKlaim process actions, the logic provides probabilistic operators which naturally express steady-state probabilities as well as probability measures of paths specified with typical until formulae. The logic integrates both the state-based paradigm and the action-based one and provides specific atomic propositions addressing data and process distribution. It also provides specific atomic propositions for actions in order to characterize relevant activities taking place during executions.

The formal semantics of $MoSL$ has been presented and a mapping from a large fragment of the logic to aCSL, the action based Continuous Stochastic Logic described in [19], has been discussed. Details on the mapping, included its formal proof of correctness are reported in [12]. The availability of such mapping(s) provides the possibility of model-checking systems modelled by StocKlaim against requirements specified in $MoSL$ using existing model-checkers for aCSL, like ETMCC. Together with M. Loreti, we are currently implementing the translation function between the two logics and the actual associations of a transition system to StocKlaim processes that will represent the model for the formulae to be checked.

The next research steps we intend to take are on one hand to complete the implementation of the above mentioned mapping, and on the other to investigate the possibility of extending ξ to deal also with formulae containing **newloc**. Our conjecture is that, under the assumption of a finite number of localities in the model—which is reasonable in the context of traditional model-checking algorithms—a formula containing **newloc** can be translated to proper disjunctions, each disjuct being obtained by substituting the free locality variables with suitable combinations over the localities. We shall also investigate feasibility and convenience to develop direct model-checking algorithms. Another issue will be the extension of StocKlaim in order to cover a larger subset of Klaim and the related extension of $MoSL$.

Finally, we shall also consider a more expressive logic that allows to reason about (spontaneous) sites failures and shall study paradigmatic examples to assess adequacy and expressiveness of the proposed logics.

Acknowledgement

This work has been partially funded by Project EU-IST IST-2001-32747 Architectures for Mobility (Agile).

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous-time Markov chains. *ACM Trans. on Computational Logic*, 1(1):162–170, 2000.

[2] C. Baier, J. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J. Baeten and S. Mauw, editors, *Concurrency Theory*, volume 1664 of *LNCS*, pages 146–162. Springer-Verlag, 1999.

[3] L. Bettini, R. De Nicola, and M. Loreti. Formalizing properties of mobile agent systems. In F. Arbab and C. Talcott, editors, *Coordination Models and Languages*, volume 2315 of *LNCS*, pages 72–87. Springer-Verlag, 2002.

[4] L. Bettini, V. Non, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim project: theory and practice. In C. Priami, editor, *Global Computing: Programming Environments, Languages, Security and Analysis of Systems*, volume 2874 of *LNCS*, pages 88–150. Springer-Verlag, 2003.

[5] L Caires and L. Cardelli. A spatial logic for concurrency (part I). *Inf. and Comp.* 186(2):194–235, 2003.

[6] L. Cardelli. A language with distributed scope. In *22nd Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 286–297. ACM, 1995.

[7] L. Cardelli. Abstractions for mobile computations. In J. Vitek and C. Jensen, editors, *Secure Internet Programming*, volume 1603 of *LNCS*, pages 51–94. Springer-Verlag, 1999.

[8] L. Cardelli and A. Gordon. Anytime, anywhere: modal logics for mobile ambients. In *27th Annual ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–377. ACM, 2000.

[9] G. Castagna and J. Vitek. Seal: A framework for secure mobile computations. In H. Bal, B. Belkhouche, and L. Cardelli, editors, *Internet Programming Languages*, volume 1686 of *LNCS*, pages 47–77. Springer-Verlag, 1999.

[10] P. D'Argenio, J. Katoen, and E. Brinksma. Specification and analysis of soft real-time systems: Quantity and quality. In *Real-Time Systems Symposium*, pages 104–114. IEEE CS Press, 1999.

[11] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Trans. on Softw. Eng.*, 24(5):315–329, 1998.

[12] R. De Nicola, J. Katoen, D. Latella, and M. Massink. Towards a logic for performance and mobility. Tech. Rep. 2005-TR-01, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2005.

[13] R. De Nicola, D. Latella, and M. Massink. Formal modeling and quantitative analysis of KLAIM-based mobile systems. In *ACM Symposium on Applied Computing (SAC).* ACM Press, 2005. Also available as Technical Report 2004-TR-25; CNR/ISTI, 2004.

[14] R. De Nicola and M. Loreti. A modal logic for mobile agents. *ACM Trans. on Computational Logic*, 5(1):79–128, 2004.

[15] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic KLAIM. In R. De Nicola, G. Ferrari, and G. Meredith, editors, *Coordination Models and Languages*, volume 2949 of *LNCS.* Springer-Verlag, 2004.

[16] G. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model-checking verification environment for mobile processes. *ACM Trans. on Software Eng. and Methodology*, 12(4):440–473, 2003.

[17] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[18] S. Hart and M. Sharir. Probabilistic temporal logics for finite and bounded models. In R. De Millo, editor, *ACM Symposium on Theory of Computing (STOC)*, pages 1–13. ACM Press, 1984.

[19] H. Hermanns, J. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods (IFM)*, volume 1945 of *LNCS*, pages 420–439. Springer-Verlag, 2000.

[20] H. Hermanns, J. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking Markov chains. *J. on Software Tools for Technology Transfer*, 4(2):153–172, 2003.

[21] V. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, 1995.

[22] S. Merz, M. Wirsing, and J. Zappe. A spatio-temporal logic for the specification and refinement of mobile systems. In M. Pezzé, editor, *Fundamental Approaches to Software Engineering (FASE)*, volume 2621 of *LNCS*, pages 87–101. Springer-Verlag, 2003.